09682576

client-side encoding

Homepage | Advanced Search

Search using: HotBot | Google | Ask Jeeves

CUSTOM WEB FILTERS Tools | HotBot Skins | P

Date: **Before September 22 2001** [Edit this Search]

SPONSORED LINKS (filters not applied)

- **Encoding** Software
  Automated **encoding** and distribution for multiple formats. Try for Free!
  www.anystream.com

WEB RESULTS (Showing Results 1 - 10 of 7,957)

1. **Client Side** State - HTTP Cookies
PERSISTENT **CLIENT** STATE. HTTP COOKIES. Preliminary Specification - Use with caution. INTRODUCTION ... scripts) can use to both store and retrieve information on the **client side** of connection .....
www.netsw.org/infosys/www/docs/cookie_spec.html - January 23, 1996 - 12 KB

2. **Client Side** State - HTTP Cookies
PERSISTENT **CLIENT** STATE. HTTP COOKIES. Preliminary Specification - Use with caution. INTRODUCTION ... scripts) can use to both store and retrieve information on the **client side** of connection .....
www.salem.mass.edu/~tevans/cookiespec.htm - December 5, 1999 - 14 KB

3. SCRIPT - **Client Side** Script
HTML Elements. SCRIPT - **Client Side** Script. The SCRIPT element includes a **client-side** script document. **Client-side** scripts allow greater interactivity in a document by responding to user e CHARSET attribute gives the character **encoding** of the external script (typically ISO ...
htmlstudio.20m.com/special/script.html - April 30, 2000 - 9 KB

4. Request **Encoding** as Objects Proxies
... object's server must also be represented on the **client side** of the Request **Encoding** as Ob connection ... important on the **client side** of all Request **Encoding** as Objects connections ...
www.casbah.org/capi/ldo.reo.proxies.html - August 1, 1999 - 11 KB

5. OOP-Reserch: My proposal to Servlet and JSP Specification - Our...
... string, we have to know the character **encoding** of the **client side**. The character array can decoded ... with the appropriate Java character **encoding**, the original string can be re ...
www.oop-reserch.com/proposal/proposal_1.html - June 17, 2000 - 9 KB

6. **Client Side** Automated Form Entry
... WD-form-filling-960416. **Client Side** Automated Form Entry ... For example people might st **encoding** their business card information into their homepage so that it can be captured ...
www.w3.org/TR/WD-form-filling.html - April 16, 1996 - 15 KB

7. Simple Script Security
... all the functions for encrypting and **encoding** web documents on **client side** and save as or files ... Decrypting is done also in **client side**, so no secret information is ...
www.cgi-network.net/links7/link_out.php?id=3917 - July 1, 2001 - 20 KB

8. **Client-Side** JavaScript Reference
New Features in this Release. Changes to the Array object. When you specify a single numeric p with the Array constructor, you specify the initial length of the array. ... See the **Client-Side**...
stefwest.d2g.com/javascript1.3 - August 5, 2000 - 35 KB

9. untitled
... and return one of serveral variants. 2)**Client-side. Client-side** content negotiation (which is http ... as a transfer or content **encoding**), please see the description of CE ...
www.srehttp.org/samples/negotiat.doc - March 27, 2001 - 42 KB

10. Common Markup for micropayment per
... Embedding information in HTML pages-**Encoding** for Plug-in or Applets ... Detecting the Brow capabilities. Generate **Encoding** using Javascript on the **client side**. List of payment systems .
www.ecash.com/MicroPayment/micropayment.htm - March 26, 2001 - 103 KB

« Previous | Next »

Search for "**client-side encoding**" using: Google, Ask Jeeves

Portions powered by Inktomi

14 September 1999

SRE-http and Content Negotiation

Content negotiation refers to the choosing of a best representation of a web resource from several alternatives. SRE-http supports several http/1.1 compliant mechanisms for performing content negotiation.

Contents:

---

1) Introduction

Content negotiation refers to the choosing of a best representation of a web resource from several alternatives.

In general, content negotiation is used to choose between resources created with one of several languages, or one of several mimetypes. Content negotiation can also be used to choose between resources using alternate character sets, and to choose shorter documents. In the future, content negotiation may also be used to choose documents using certain features (such as documents using different versions of html).

There are basically two forms of content negotiation:

1) Server-side.
   Server-side content negotiation (which is defined in http/1.0) is performed by the server -- the server uses ACCEPT request headers (provided by the client) to automatically choose and return one of serveral variants.

2) Client-side.
   Client-side content negotiation (which is new to http/1.1) is accomplished by the client automatically requesting one of several variants:
   the choice is based on a "variants list" (that contains URI's and descriptive information) contained in response headers returned by the server.

SRE-http supports both server-side and client-side content negotiation.

II) Specifying a negotiable resource

To specify a "negotiable" resource (a web resource with several possible variants) requires 3 steps:

1) Create several "variants" of a resource
2) Create a file, in your web-space, containing a "variant list"
3) Create a special NEGOTIATE alias pointing to this file

These steps are described in the next three sections.

Notes:

* SRE-http's implementation of content negotiation is loosely based on Apache 1.3. You might want to examine
  http://www.apache.org/docs/content-negotiation.html
  for a different description.

* For a technical discussion of content-negotiation, see RFC 2295 "Transparent Content Negotiation in HTTP" at http://gewis.win.tue.nl/~koen/conneg/

* The rationale for client-side content negotiation is that the browser is best equipped to choose a variant (given descriptive information on the variants mimetype, language, etc.) Furthermore, it is thought that it is wasteful of bandwidth to provide the full range of Accept: headers on every request (since most resources will not be subject to content negotiation). However, client-side content negotiation does require two round trips (one to get an alternates list, and a second to get the desired variant).

* If all you are interested in is GZIPping a response (as a transfer or content encoding), please see the description of CE_GZIP (in INITFILT.DOC).

II.1) Creating variants

Basically, a variant is any server resource. This includes documents, images, and even cgi-bin scripts and sre-http addons. The notion is that variants all represent variations of the same information. For example, you may have several translations of the same document (say, an English, Finnish and Korean version) which you'ld like to automatically send to the appropropriate clients.

There are a few constraints:

1) variants must be on the same server as the "variant list" (discussed next). In addition SRE-http enforces the "good practice" (from a security standpoint) of requiring that each variant to be in (or under) the same directory as the variant list.
2) variants must be retrievable via GET requests. That is, each variant should be accessible with a standard URL; which also means that

content negotiation will NOT work with POST requests.

II.2) Creating a Variant List

The variant list is at the heart of SRE-http's implementation of content negotiation. The variant list is a simple (text) file containing several multi-line records. This file should be accessible from the web.
That is: the variant list must be in (or under) the GoServe data directory, or in an SRE-http virtual directory.

Each record in the variant lists must specify a URI (a selector), and several pieces of identifying information. This identifying information is used to specify up to six "dimensions of negotiation": such as the mimetype, charset, language, encoding, features, and length.

The syntax of these multi-line records is (note that a wildcarded form of these records can also be specified, as discussed in section II.4 below):

```
URI: a_selector
Content-type: type/subtype ; charset=a_charset ; qs=m.mm
Content-language: l1, l2
Content-encoding: enctype
Content-length: nnnn
Option: an option
```

Where:

URI is required.

URI should be a valid selector; that is, site information can not be included -- the resource must be on the same site as the variant list. Note that relative URI's are interepreted relative to the location of the "variant list".
** Since variants (as identified by these URI entries) MUST be in (or under) the directory containing the variant list file, use of "relative" uri's is recommended.

Content-type: Content-type is required. It contains 3 sub-fields.

type/subtype: Only the type/subtype subfield is required. It identifies the mime-type of this variant.

charset: Optional. Identifies the character set. If not specified, a ISO-8859-1 (latin1) is assumed

qs: Optional. The "selection quality". Must be between 0.0 and 1.0. 0.0 means "unacceptable", 1.0 means "perfect representation". If not specified, a value of 1.0 is assumed. All else equal, variants with higher values are preferentially chosen.

Content-language: Optional.

A comma delimited list of 2 character languages codes (i.e.; EN for English, FR for French, DE for German). Note that the 2-2 letter codes, such as En-US, are

shortened (only the first two characters are used).

Content-encoding: Optional.

A comma delimited list of content encoding types (i.e.; identity, gzip, and compress).

Note: if a variant with GZIP content-encoding is chosen, then on-the-fly GZIP content encoding (as controlled by the CE_GZIP parameter in INIT_STA.80) will be suppressed.

Content-length: Optional.

The length of the resource. If not specified, a length of 0 is assumed. Note that "longer" resources are less likely to be chosen (all else equal).

Description: Optional
An optional description of this variant.

Features: Optional
Features allows you to note special "features" of the variant. Although not widely supported, in the future browsers may use such information to choose a variant (SRE-http does NOT use features in its default variant selection algorithim).
For details on "feature", please see RFC2295.

Option: Optional
You can specify "selector specific" advanced options to be assign to this variant. These complement selector specfic advanced options that are assigned to the original selector (that were assigned to the selector that points to the variant list).
You can have multiple Option: lines in a given entry.
Note that Option: entries are NOT used by SRE-http's default variant selection algorithim.

Notes:

* The "qs" quality specified in content-type applies to the entire variant. You should NOT enter "quality" terms for the other content- factors.

* Appendix A contains an example of a variant list file.

* When a variant is selected, its Content-Type, Content-Language, and Content-Encoding are returned as response headers. Thus, in most cases the Content-Type specified in a variant list will override the "default" Content-Type (i.e.; the mimetype derived from the file's extension). The exception to this rule is when a mime-type is specified as an advanced option, or when the variant is a CGI script or an SRE-http addon (in which case the script/addon should provide the content-type information).

* If the last variant in a variant only has a URI: field, then it is treated as a "fall back" variant, and will (typically) be used only if there is no best match.

II.3) Identifying a negotiable resource.

SRE-http uses a special "alias" to identify variant lists.
Unless an alias is used to explicitily identify a variant list, a
request for a variant list will be treated in the normal fashion.
That is, the variant list file would be returned verbatim (say, as a
text/plain response).

To identify a variant list, you can either:

 a) add an entry to to your ALIASES file, using:
        sel   !NEGOTIATE target
 b) define a realm in ATTRIBS.CFG using:
       rule: sel
       redirect: NEGOTIATE= target

where:
     sel = a selector that points to identified as a negotiable resource.
            Alternatively, you can use a wildcarded selector.
  target = optional -- either a fully qualified file name or a
            selector.

Typically, target is not specified; in which case SRE-http use its
normal rules to map sel to a file containing a variant list
(that is, sel is assumed to be under the GoServe data directory
or under a virtual directory).

If specified, target is used as the variant list. If target
is not a fully qualified file name, it should be relative
to the GoServe data directory (or to a virtual directory).

In all cases, the variants must be "neighboring", which means they
MUST be relative to the directory that contains the variant list.

**   If you use a wildcarded selector (containing *), you MUST specify
**   a target -- see the section II.3.a for the details.

Some examples:

  ALIASES.IN example:
     /VARTEST/VAR1.LST !NEGOTIATE

  ATTRIBS.CFG example:
     realm: negot1
     rule: /VARTEST/VAR1.LST
     redirect: Negotiate=

  Assuming that your GoServe data directory is E:\WWW, this could mean:
     /VARTEST/VAR1.LST is a negotiable resource, with a variant list defined
     in E:\WWW\VARTEST\VAR1.LST.

Notes:
  * As noted, "resources" pointed to by a variant list must be
    "neighboring"; they must be in or under the same directory as
    the variant list file.

* Unlike most SRE-http aliases, there is no explicit "replacement". Actually, you can think of the variant list itself as an extension of SRE-http's aliasing -- it contains instructions used to decide which (of several) replacements to use.

II.3.a) Wildcarded form

When sel contains a wildcard (an *), you MUST use target to specify the file containing the variant list (you can use either a fully qualified file name, or another selector).

For example, using ALIASES.IN:

```
     /MANUALS/*.HTM !NEGOTIATE /MANUALS/DOCS.LST
```

or, using ATTRIB.CFG

```
     Realm: negot2
     Rule: /MANUALS/*.HTM
     Redirect: Negotiate=/MANUALS/DOCS.LST
```

In this case, all request selectors that match MANUALS/*.HTM (note that the leading / is ignored) will use the variant list specified in /MANUALS/DOCS.LST. Please see the next section for details on how variants are resolved when this wild_sel form is used.

Once you've accomplished these steps, all you need to do is put a URL pointing to sel (or that will wildcard-match sel); and hope that your client's browser either provides useful ACCEPT headers, or knows how to do client side content negotiation.

II.4) Wildcarded variants

As outlined above, one creates a unique variant list (and a unique alias) for all negotiable resources. This may become quite tedious, especially when you have multiple sets of documents. For example, if you have a 10 chapter manual in 3 languages (hence, 30 files), it could be advantageous (that is, a lot less trouble) to use some wildcarded "variant list" for all 10 chapters.

In recognition of this possibility, SRE-http supports a special form of variant list that supports such "multiple sets of negotiable resources". The specification of these sets requires two changes to the simple case.

The first difference is discussed above -- the use of the "wildcarded form" of sel. The second involves modifications to the variant list file.

Recollect that the wildcarded form directs many possible "request selectors" to a single variant list. Thus, the variant list should contain information that allows the request selector to influence the value of the URI: field of each record in the variant list. To do this, two steps are required.

a) A (case insensitive)

```
    PATTERN: wild_sel
```

entry should be put at the top of the variant list.

For example:
Pattern: /manuals/*.HTM -- (note that a leading / is ignored)

** The value of "wild_sel" used in a PATTERN: entry should be the same as the "wild_sel" used as the "target" portion of the wildcarded alias (that is used to identify the variant list).

b) The URI: fields may contain * characters.
SRE-http will replace these * (in the URI: entries) with corresponding portions of the request selector.

For example:
i) the request selector is: manual/chap1.htm
ii) the alias is: manual/* !negotiate manual/docs.lst
iii) manual/docs.lst contains

```
pattern: manual/*

URI: de/*
Content-type: text/html
Content-Language: de

Uri: en/*
Content-type: text/html
Content-language: en
```

then
a) If the request contain an accept-language: en request header, then en/chap1.htm would be used

b) If the request contain an accept-language: de request header, then de/chap1.htm would be used

That is, the * in the Pattern: (in manual/*) "corresponds to" chap1.htm, which is then used as a substitute for the * in the various URI: entries.

III) The content negotiation algorithim.

The following sketches the default content negotiation algorithim used by SRE-http. Note that this is used both for "server side negotiation" (when the client does not include a Negotiate: request header), and as a "remove variant selection algorithim" (when the client includes a Negotiate: * request header).

Alternatively, you can specify your own "server side "content negotiation algorithim on a selector specific basis -- see section III.a for the details!

1) First, SRE-http checks for a "Negotiate:" request header. If no such header exists, then server-side negotiation is always attempted. If this header does exist, then server-side negotiation may be attempted (see the notes for details).

2) If server-side negotiation is to be attempted, by default

the following selection algorithim is used.

If the client allows a "remote variant selection algorithim" (by including a Negotiate: n.n request header), then a custom procedure can be used instead (see Appendix B for the details).

Note that this is a "leave as soon as a definitive answer is found" method -- latter steps are only used if earlier steps yield ties. Furthermore, variants eliminated in earlier steps are NOT available -- they are NOT considered in latter steps. Lastly, if all variants are eliminated, a suitable "could not find representation" response is immediately returned.

a) Accept: headers are read. Accept: headers contains information on acceptable mime-types. This information can contain "selection quality" (q) information.
The variant with the best "combined" quality is used. Combined quality is deterimined by multiplying the variant-list qs (quality) by the accept: header "q: factors".
If there are ties (i.e.; several mimetypes have a combined quality of 1.0), then move to step b. If there is no accept: header (most browsers send some form of Accept: header) then skip this step.

b) Accept-language: headers are read. These request headers can contain "language specific q modifiers". The variant (of those surviving step a) with the highest language "q" factor is used.
If there are ties, move to step c. If there is no accept-language header, this step is skipped. Note that the content-language entries in the variant list should NOT include "q" factors.

c) Accept-Encoding: headers are read. These headers can also contain "encoding specific q modifiers". The variant (of those surviving step b) with the highest encoding "q" factor is used.
If there are ties, move to step d. If there is no accept-encoding header, this step is skipped. Note that the content-encoding entries in the variant list should NOT include "q" factors.

d) Accept-charset: headers are read. Variants that do not match this charset are dropped.
If there are ties then move to step e. If there is no accept-charset header then skip this step.

e) Use the variant with the smallest content-length (as pulled from the variant list).
If there are ties, move to step f.

f) Use the first of the remaining variants.

Note:
If all variants are removed at any step in this process (say, no variants have a content-language, and an explicit Accept-language was specified), then the algorithim exits.
In this case, if a default variant was specified, it will be used.
Otherwise, either a 406 (for http/1.1 clients) or a 404 (for http/1.0 clients)
response is returned.

3) If client-side negotiation is to be used, SRE-http returns a special "300" return code. The body of the response contains an <UL> list containing links to each variant. Furthermore, the variant list (suitably formatted) is returned in an Alternates: response header.

Note:

* SRE-http recognizes several values in the Negotiate: header (any combination of them can appear in a comma delimited list)

```
   trans:  client supports client side content negotiation
   vlist:  always send Alternates: response header (implies trans)
       *:  server should attempt to choose best variant (implies trans).
guess-small: always send Alternates: response header (implies trans)
             This is minimal support -- with full support, sometimes
             a sufficiently small "best variant" guess is returned.
   n.n     : A decimal number, such as 4.2.
             This specifies the major and minor versions of a customized
             "remote variant selection algorithim" (RVSA) -- a custom
             procedure to replace SRE-http's built in variant selection
             algorithim. See APPENDIX B for details on how to
             add a procedure to implement your favorite RVSA!
```

In the following cases, server-side negotiation is not attempted:

```
Negotiate: trans
Negotiate: vlist
Negotiate: vlist,trans
Negotiate: guess-small
```

In these cases,SRE-http returns a "300 Multiple Choices" response.

A "300" response always includes an Alternates: header (containing a suitably formatted version of the variant list). In addition, a <UL> list of links to each variant is returned in the body of the response.
The assumption is that (in most cases) the user-agent (the browser) will use the variant list to choose a best variant; but if it can't, the list will be displayed and the human can manually choose.

When * is included, then server-side content negotiation will be attempted. In particular:
Negotiate: *, vlist
means "attempt to find best match, but always return the variant list"
In contrast
Negotiate: *
means "return the best match, only return a variant list if no best match found".

When a "n.n" RVSA token is included, and the appropriate procedure has been defined, then SRE-http will attempt to use the indicated RVSA algorithim for server-side content negotation.
See Appendix B for the details.

In either case (that is, whenever * or an n.n appears in the Negotiate: header), failing to find a best match causes a "406 No acceptable representation" response to be returned. This 406 response is similar to the 300 response described above -- it contains an Alternates: header and a <UL> list of links.

* To repeat: if no Negotiate: request header exists, then server-side content negotiation is attempted. If no best-match can be found, then a 404 response, containing a <UL> list of links to each variant, is returned (but no Alternates: header).

* Whenever server-side negotiation (either pure server side, or due to a Negotiate:* header) is succesful, a Vary: header will be included (Vary headers are used by proxy servers).

* When a quality value is not specified (either as qs factor in the Content-type field in the variant list, or as a "q" factor in an Accept: or Accept-Language request header), a value of 1.0 is assumed. There are a few exceptions:
  i) if a */* or xxx/* appear (in the Accept: header), and if no other mimetypes have a q modifier, then */* is assigned a value of 0.01, and xxx/* is given a value of 0.02.
  ii) records in the variant list with no content-language field are given a language quality factor of 0.01.

  Both these exceptions are tricks that cause these "defaults" to be used when no better match exists.

* You can specify multiple languages in the Content-language entry (in a variant list record). However, you can NOT specify a "q" modifier for these languages. Note that the accept-language header MAY contain several languages, each of which may have a "q" modifier.

* If no charset is specified (in a variant list record), then ISO-8859-1 (latin1) is assumed.

* If no content-length is specified (in a variant list record), then a length of 0 is assumed. That is, content-length is really a final "quality check", with lower values (and unspecified values) preferred.

* Hint: the Options-General-Language tab of Netscape can be used to automatically generate Accept-language headers.

* WARNING: the variant returned by server side negotiation should NEVER be a negotiable resource. If this should happen, SRE-http will return a 506 response (Variant Also Negotiates).

III.a) Using your own server-side content negotiation algorithims

Although client-side negotiation can be quite powerful, it does require fairly advanced http/1.1 clients. Until this is commonplace, server-side negotiation is a good alternative.

Although the Apache-emulating content negotation algorithim

(described above) is fairly powerful, it is limited. For example, there is no way that cookies can influence the choice of variant. Therefore, SRE-http allows you to specify your own "server side" content negotiation algorithims, on a selector specific basis.

To do this, you must:

a) Include a
   PROC: procname
   element at the top of your variant list file.

b) Create a procedure with "procname" that can choose a variant. This procedure should reside in a .CMD file in the GoServe working directory, or in macrospace.

For example, your variant list could look like:

```
URI: bar
Proc: SREF_NEGOTIATE_1

Uri:bar.html
content-type: text/html ; q=1.0

Uri:bar.txt
content-type: text/plain ; q=0.7
```

which would mean "use the SREF_NEGOTIATE_1 procedure to choose between these two variants".

III.a.i) The Custom Procedure

Your custom procedure will be passed four arguments:
i) the original request selector,
ii) the variant list file (that the original selector maps to)
iii) number of entries in the variant list
iv) the variant list

For example, you can read these arguments using:

```
parse arg  sel0,afile,nentries,varilist
```

Varilist (the variants list) will have the form:

```
n
spec_1
 ...
spec_n
```

where:
n = # of variants
spec_j = specifications for alternate j, with structure:
"a_sel" q {field1 values1} {field2 values2} ....

For example:

```
4
"tst.1" 1 {type text/plain } {language en } {charset cyrillic }
"tst.2" 0.3 {type text/plain } {language fr } {features tables def}
"tst.3" 1 {type text/html } {language en } {length 20065 } {charset latin5
}
```

```
    "/status?" 1 {type application/octet-stream } {language gr }

  Note that each entry appears on a single line (which may be quite long).


The procedure should return:
    statcode  header_list
where
 statcode:
       '' -- a blank line means "could not find a best match,
             use SRE-http's default algorithim instead"
       0 -- 0 means "could not find best match, return a 406 response"
       j -- A number between 1 and n, which points to the best variant.

 header_list (optional):

       only used if statcode=j (j=1..n). This should contain a space delimited
       list of the request headers used to choose the variant. This list
       is returned as a Vary: response header, and will then be used by
       proxy servers to determine the cacachability of this variant.
       If you don't specify header_list, a Vary: * header will be
       added (which implies that this variant is not cachable).

Example of a return:
    return '3 Accept Accept-Language '


Notes:
  *  '' and 0 differ in that a return of 0 causes SRE-http to stop trying
     to find a best variant (and return a 406 response); wheras as ''
     means "now try the default algorithim" (which might yield a
     match).

  * Example of a header_list:  If you used the ACCEPT and ACCEPT-LANGUAGE
     request headers to determine the best variant, header_list should
     be "ACCEPT ACCEPT_LANGUAGE" (without the quotes). Or, if you also
     used a cookie, then header_list should be
     "ACCEPT, ACCEPT_LANGUAGE, COOKIE".

  * The NEGOT.CMD file, that comes with SRE-http, provides a well-documented
     example; it basically emulates the default "server side" content
     negotiation algorithim.

III.a.ii) Special procnames

Since you can have several different procedures, for several different
negotiable resources, you can use any "procname" you wish -- so long
as it is a legal REXX procedure name, and so long as it can
either be found in macrospace or in the GoServe working directory.

However, since this flexibility comes at some cost (SRE-http has to
use the somewhat slow INTERPRET command), it is recommended that
one of the following procnames be used:

  SREF_NEGOTIATE_1
  SREF_NEGOTIATE_2
  SREF_NEGOTIATE_3
```

NEGOT1
NEGOT2
NEGOT3

SRE-http is coded to recognize these names, thereby avoiding the use of INTERPRET. In particular, the SREF_ names should refer to procedures that you load into macrospace (say, using the CUSTOM_INITS facility of SRE-http), and the NEGOT names should refer to .CMD files in the GoServe working directory.

In any case, the procedure will be passed the arguments described above, and must return either a 0 (no best variant found), or an integer pointing to the best of the n variants provided.

III.a.iii) Useful procedures.

In order to ease the task of writing a custom procedure SRE-http provides two procedures that you can use to search and extract information from the the variant list: SREF_NEGOTIATE_SEARCH and SREF_NEGOTIATE_EXTRACT. See Appendix D for the details.

In addition:

* the GoServe reqfield(header_name) function can be used to lookup other specific request headers (see GOSERVE.DOC for the details).
* The SRE-http SREF_GET_COOKIE(cookie_name) to read the value of the cookie_name cookie (see SREFPRC.DOC for the details).

For an example of how to a custom "server side" content negotiation procedure, please see NEGOT.CMD.

Appendix A) Sample Variants file.

---------- Start example ----------

```
; this is sample variant file

URI: foo
PROXY-RVSA: 1.3

Uri:foo.fr.de.html
content-type: text/html
content-language: sp,fr-ca,de
content-length: 2005

Uri:foo.txt
content-type: text/plain ; q=0.5
content-length: 2005
content-language: en
description: this is the english text version
```

```
Uri:foo.gz
content-type: text/plain ; q=0.5
content-length: 2005
content-encoding: gzip
content-language: en
option: header add x-advisory: needs Gzip
option: header add set-cookie: foobar=this is foobar

uri: /status?
content-type: application/octet-stream

uri: foo.en.html
content-type: text/html ; charset=iso-8859-1 ; q=1
content-language: en
features: tables frames

uri:foo.def0
```

---------- End of example ----------

Notes:

* Lines beginning with ; are comments, and are ignored

* Each, typically multi-line record, contains a
  URI: entry, and (optionally) a combination of content-type,
  content-language, and content-length entries.

* Blank lines are treated as "record delimiters"

* The first "one line record" URI: entry is optional -- it's skipped

* An (optional) Proxy-RVSA: entry (at the top of the file,
  before the start of the first non-one line record) is used to signal
  to proxies what RVSAs they can use.

* Note that PROC:, if used, should also be placed before the
  start of the first record (i.e.; next to the first "one line record" URI).

* Content-length is the estimated size of the file,
  it does NOT have to be the actual size of the file.
  All else equal, smaller "lengths" are preferentially returned.

* Content-language is a comma delimited set of languages

* Content-encoding is an encoding type, with "identity" meaning
  "no encoding". Only one encoding-type can be specified.

* Content-type contains 3 fields. The type/subtype field is required,
  the charset= field is optional (iso-8859-1 is the default), and qs
  is the (optional) "quality" measure used to weight the variants.

* Note the last entry is a "fall back" variant (the client's user-agent
  can choose this if all else fails).

Appendix B: Specifying a Remote Variant Selection Algorithim

SRE-http provides a simple hook by which a custom remote variant selection algorithims (RVSA) can be implemented. The process is:

a) obtain the RVSA.

b) write a rexx procedure that calls this rvsa

c) save this procedure into macrospace using a name of SREF_RVSA_n, where "n" is the major version number of the rvsa.

   For example, if the request contains:
       Negotiate: 2.3,vlist
   then SREF_RVSA_2 is appropriate "macrospace" procedure name.

In the simplest case, the rvsa procedure will be a rexx procedure; in which step a and b are combined.

The rvsa procedure (say, SREF_RVSA_2) will be called with two arguments, the version number, and the list of alternates. For example, to read these arguments the procedure could use:

```
parse arg version,altlist
```

In the above example (Negotiate: 2.3,vlist) the version number would be "2.3" (without the quotes). The list of alternates is simply a copy of the Alternates: header (that would be returned to the client).

The rvsa procedure should use this information, along with request specific information (such as the various Accept headers, which can be read using the GoServe reqfield function) to determine which variant is best. The number of this best variant should be returned; or a 0 should be returned if there is no best match.

For example, if the second variant (in the list of alternates) is best, then SREF_RVSA_2 should return a "2" (without the quotes).

Notes:

* You can use RXU, REXXLIB, or other DLLs to create macrospace procedures

* For details on the structure of the alternates list, see RFC2295. (see Appendix 3 for an example).

Appendix C: Example of a TCN Request and Response

The following is a simple example of TCN request, and the response from SRE-http.

Assuming that:
  TSTHTM/TSTHTM.NEG
is a negotiable resource, and TSTHTM.NEG has the following structure:

```
   ;---- TSTHTM.NEG is a negotiable resources
   uri:tsthtm

   uri:tst.1
   content-type: text/plain; qs=0.8
   content-language: en

   uri: tst.2
   content-type: text/plain ; qs=0.3
   content-language: fr
   description: The French Version
   features: tables [abc def]

   uri: /gene_test?
   content-type: application/octet-stream; charset=cyrillic
   content-language: ru
   ;---- End of TSTHTM.NEG
```

Requesting a url of:
  http://foo.bar.net/tsthtm/tsthtm.neg

causes a browser to issue the following request:

```
   GET /tsthtm/tsthtm.neg  HTTP/1.1
   HOST:foo.bar.net
   Negotiate: vlist,*
   Accept: text/plain
   Accept-language: fr
```

This would yield the response:

```
  HTTP/1.1 200 Ok
  Date: Sat, 26 Jun 1999 13:49:10 GMT
  Accept-Ranges: bytes
  Connection: close
  Last-Modified: Thu, 24 Jun 1999 05:52:38 GMT
  ETag: "990624015238_18;7964E38D"
  Server: GoServe for OS/2, version 2.52; SRE-http 130607.1
  Content-Type: text/plain
  Tcn: choice
  Content-location: tsthtm/tst.1
  Vary: negotiate,accept
  Alternates: {"tst.1" 0.8 {type text/plain } {language en }},
  {"tst.2" 0.3 {type text/plain } {language fr } {features tables [abc def] }
  {description "The French Version"}},
  {"/gene_test?" 1 {type application/octet-stream } {language ru }
  {charset cyrillic }}
  Content-Language: en
  Content-length: 18
  Cache-control: public
```

Notes:
  * Since * appears in the Negotiate: header, SRE-http will
    attempt the default server side negotiation. The Accept: text/plain
    is the only information used, and the explicit q=0.8 of the first
    entry beats the explicit 0.3 of the second entry. The implicit
    q=1.0 of the third entry is irrelevant, since the third entry's
    content-type is not text/plain
  * the Alternates: header has been reformatted (it is actually on one
    long line).
  * The contents of tsthtm/tst.1, which (in this example) has a size of
    18 characters, is sent as the body of the response.

Alternatively, if the client does not want the server to resolve the
variant:

```
   GET /tsthtm/tsthtm.neg  HTTP/1.1
   HOST:foo.bar.net
   Negotiate: vlist
   Accept: text/plain
   Accept-language: fr
```

which yields:

```
   HTTP/1.1 300 Multiple Choices
   Date: Sat, 26 Jun 1999 14:01:20 GMT
   Server: GoServe/2.52 ;130607.1
   Tcn: list
   Etag: 990626095912_324;7964E38D
   Vary: *
   Alternates: {"tst.1" 0.8 {type text/plain } {language en }},
   {"tst.2" 0.3 {type text/plain } {language fr } {features tables [abc def] }
   {description "The French Version"}},
   {"/gene_test?" 1 {type application/octet-stream } {language ru }
   {charset cyrillic }}
   Cache-Control: public
```

Note that the portion of the etag following the ; (the 7964E38D)
is the same for both requests -- it's the "variant list validator".
And since the Alternates: header is the same in both responses,
this "variant list validator" is also the same.

Appendix D: The SREF_NEGOTIATE procedures:

For both these procedures, afield can take the following values:

```
  SEL -- the variant's selector
     Q --  the quality rating
    TYPE -- content-type
  LANGUAGE -- comma delimited list of languages
    CHARSET -- the character set
    ENCODING -- the content encoding
    FEATURES -- Features.  Can be used for miscellaneous features
  DESCRIPTION -- Description of variant
```

SREF_NEGOTIATE_SEARCH:

Syntax:
   matchlist=SREF_NEGOTIATE_SEARCH(avalue,afield,varilist,best,onlyin)

where:
    avalue : a value, or a set of values, to examine
    afield : the name of the "field" in the variant list, values
             from this field will be compared against values in
             avalue.
  varilist : the "variant list" (as provided by sre-http)
    best   : optional. If 1 or 2, then do a "quality weighted" best
             match, and return only the top scoring variant
             (or variants, if a tie for top)
   onlyin : optional. A index of the entries in the variant
             list to examine.

   matchlist: a space delimited list of integers

Description:

Matchlist is an index to the variant list -- the indicated
variants have a value in "afield" that is a match (or a best
match) to a value in "avalue".

If no matches, returns a 0.

Basically, this procedure eases the comparison of Accept- headers
against the variant list; a typical use is to read an
accept- header, and use its value as "avalue".

Values in "avalue" can contain * wildcards. However, values in
"afield" should NOT contain * characters.

For both avalue and afield, values are seperated by commas.
Values in "avalue" may also be "modified" by ;q=n.nn -- these modifiers
are used as the "weighting factor" for that value. Thus,
different values within "avalue" can have different weights.

If an "afield" is missing from a variant, it is skipped.
Note that the "language" and "features" fields may contain
more then one value; all other fields contain 1 value (or
0 values, if they are not defined).

If best=1, then instead of returning all variants that "match", only
return the variants with the highest match (if a tie for highest, return
all the top matches).

The "highest match" is determined by using the "q" weighting factors that
may appear in "avalue". Note if a value in avalue does not have an
explicit q factor:
   a) if it does not contain a *, q=1 is used.
   b) if it has one * (say, text/*), q=0.02

c) if it has two * (say, */*), q=0.01

If best=2, then modify this by multiplying the q weighting factor of the value from "avalue" by the "Q" field of the variant.

If onlyin is specified, then only the records indexed by onlyin are searched (records not indexed by onlyin can NEVER appear in matchlist).

Examples:

```
kth=sref_negotiate_search('fr;q=0.5, sp','language',varilist,1)
mth=sref_negotiate_search('text/plain','type',varilist,2,'1 3 6 7')
mth=sref_negotiate_search('text/plain, text/* , */* ','type',varilist)
```

SREF_NEGOTIATE_EXTRACT:

Syntax:

```
avalue=SREF_NEGOTIATE_EXTRACT(afield,j,varilist)
```

where:

```
afield = a "field name"
j      = an index into the variant list
varilist = the variant list
```

Description:

Extract the value of the "afield" field from the j'th entry in the variant list. If no such field, or if j is invalid (less then 1 or greater then the number of variants, then return '')

Alternatively, j can be a space delimited list of entry numbers, in which case avalue will be a CRLF ('0d0a'x) delimited list of the values of afield from these entries.

Lastly, if j='*', the values of afield for all entries will be returned in a CRLF delimited list.

Examples:

```
alang=sref_negotiate_extract('language',2,varilist)
atype=sref_negotiate_extract('type',3,varilist)
sels=sref_negotiate_extract('sel','1 4',varilist)
```